

## Introduction

Machine Learning 2019  
mlvu.github.io

## machine learning

### part 1:

What is Machine Learning?

Supervised ML: Classification

### part 2:

Supervised ML: Regression

Unsupervised ML

What isn't Machine Learning?

Machine learning has a very broad definition, but in practice we usually mean something very specific when we talk about it. I will attempt to describe what people usually mean by machine learning and along the way introduce the most important concepts.

In the second part we will discuss the role that search plays in machine learning, and we will delve more deeply into a specific family of ML models, called neural networks.

## your grade

### exam

### project

minimum grade 5.5	minimum grade 4.5
subject: Lectures, Literature	groups of 5
~40 multiple choice q.	subject free see Canvas for suggestions
7 homework sessions (weekly)	project sessions (biweekly)

attendance is only obligatory at the project sessions (at least one group member)

3

## reading

Reader: see links in syllabus

Meant to *support* the lectures. Almost all exam questions will be about the slides.

The literature for the course is a reader.

You may find the occasional reference to Peter Flach's book "Machine Learning". This was last year's book. We don't use it any more.



4

Covers slides, supported by literature

~40 multiple choice questions in three categories  
recall  
applied knowledge  
active knowledge (calculating stuff)

Cheat sheet available for formulas

Practice exams available

## first homework next wednesday

thing by heart, but it shouldn't take you too long to look the answers up.

### 1 Linear Algebra

In all homework and lectures, bold lowercase letters like  $\mathbf{x}$  indicate a vector, bold uppercase letters like  $\mathbf{W}$  indicate a matrix and non-bold lowercase letters like  $x$  indicate a scalar (that is, a number).

#### Exercise 1

Explain in words what the following notations represent:

1.  $f: \mathbb{R}^3 \rightarrow \mathbb{R}^2$
2.  $\mathbf{y} = \mathbf{W}\mathbf{x}$
3.  $\mathbf{z} = \mathbf{y}^\top \mathbf{x}$
4.  $\mathbf{W} \in \mathbb{R}^{5 \times 4}$

Hint: if you're not sure, see if you can find the symbols in this page:  
[https://en.wikipedia.org/wiki/List\\_of\\_mathematical\\_symbols](https://en.wikipedia.org/wiki/List_of_mathematical_symbols). Even

This is the first homework. If you find these symbols intimidating, or difficult to read, **make sure to show up to the homework session**. The mathematics of machine learning is relatively lightweight, but you do need to get used to the notation. If you have trouble with this sort of thing, the best thing to do is to face it early on and to ask us for help.

$$\begin{aligned}
 \frac{\partial \text{loss}(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} &= \frac{\partial \frac{1}{n} \sum_i (\mathbf{w}x_i + \mathbf{b} - y_i)^2}{\partial \mathbf{w}} \\
 &= \frac{1}{n} \sum_i \frac{\partial (\mathbf{w}x_i + \mathbf{b} - y_i)^2}{\partial \mathbf{w}} \\
 &= \frac{1}{n} \sum_i \frac{\partial (\mathbf{w}x_i + \mathbf{b} - y_i)^2}{\partial (\mathbf{w}x_i + \mathbf{b} - y_i)} \frac{\partial (\mathbf{w}x_i + \mathbf{b} - y_i)}{\partial \mathbf{w}} \\
 &= \frac{2}{n} \sum_i (\mathbf{w}x_i + \mathbf{b} - y_i)x_i \\
 \frac{\partial \text{loss}(\mathbf{w}, \mathbf{b})}{\partial \mathbf{b}} &= \frac{\partial \frac{1}{n} \sum_i (\mathbf{w}x_i + \mathbf{b} - y_i)^2}{\partial \mathbf{b}} \\
 &= \frac{2}{n} \sum_i (\mathbf{w}x_i + \mathbf{b} - y_i)
 \end{aligned}$$

This is one of the slides in next Thursday's lecture. For some of you this may look trivial, and for some it may look scary. You don't necessarily have to understand all such slides to pass the exam, but you do if you want to really understand machine learning.

The good news is that the math isn't as scary as it looks. You just need to get comfortable with a few basic ingredients. The first homework should help you do this. If you want to make the rest of the lectures easier to follow, I strongly recommend spending some time with the first homework.

## project

Until March 1

- EXPLORE
- Choice of software is free. Worksheets for **Python** stack.

On March 1: pick a topic

After March 1

- Do experiments, write report.

## project sessions

groups 1-30: **14 Feb, 28 Feb, 14 March**

groups 31-60: **21 Feb, 7 March, 21 March**

Informal presentations.

**Report on your progress, or lack thereof.**

Describe what problems you've run into. Discuss solutions with the group, and the TA.

Discuss what subjects you're considering.

9



Some people don't like working in groups, so they turn a project assignment into five individual assignments by immediately breaking up the work and never meeting again. Not only is this not the point of group work, it's very likely to fail: if one group member doesn't deliver, or misunderstood the idea, the whole project goes wrong.

## a word on group dynamics

- **don't:** divide duties, split up, and never meet again
- **do:** discuss level of ambition
- **don't:** pick a topic right away
- **do:** spend time to make sure you're on the same page
- **don't:** worry too much about equal workloads
- **do:** each do all the worksheets individually

We have a very diverse group of students, and many will be grouped with people you haven't met before. Make sure that you're on the same page about the kind of project you'd like to do and how complicated you want to make stuff. Nothing kills a project faster than a single highly motivated student dragging everybody into a hugely ambitious project.

To make sure that everybody is clear about the group's goals, take your time before you pick the topic. Meet every week and do lots of exploratory work. give everybody time to learn what machine learning is about, and to join the discussion. It helps if everybody does the worksheets on their own, so you have a common experience to build on.

11

## first worksheet

The screenshot shows a Jupyter Notebook interface with the title "first worksheet". The notebook is titled "Worksheet 1, Numpy and Matplotlib" and has a note: "Last Checkpoint: Yesterday at 9:42 AM (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Help, and Python 3. A CellToolbar is also visible. The main content area contains instructions for setting up the environment, a warning about re-running cells, and information about Numpy. A code cell at the bottom imports numpy as np. A note says: "If you see something like "In [1]:" next to the cell above, execution has finished."

If you're looking at this file on GitHub, make sure to follow the [instructions for getting set up](#) first. You should download these files to your machine and run them locally.

**Worksheet 1: Numpy and Matplotlib**

This is a Jupyter notebook. It consists of a series of cells. Some contain simple text, like this one, and some contain code, like the one below. Read each cell carefully. If it contains code, you can click the "play" button above to execute it. Don't execute a new cell until all previous ones have finished executing.

**NB:** If you go back up an re-run a cell, it can happen that it doesn't work anymore, because the variable names have been re-used. If something doesn't work, for any reason, please try "Kernel > Restart & Clear Output" and start again from the top. If the error keeps happening, please post a question on the Canvas message board.

We will assume that you know the basics of Python. If you don't, please work your way through [this tutorial](#) first to get caught up.

The next cell imports the library numpy (short for numeric python).

In [ ]: `import numpy as np`

If you see something like "In [1]:" next to the cell above, execution has finished.

**Numpy**

Numpy is a *Linear Algebra* library. It allows you to represent, vectors matrices and tensors as Python objects, and to manipulate these in all the ways you'd expect.

[Home](#)[Syllabus](#)[Pages](#)[Assignments](#)[People](#)[Discussions](#)[Grades](#)

# Course syllabus

Machine Learning is the practice of c  
course, you will learn the basic princi

## Course requirements

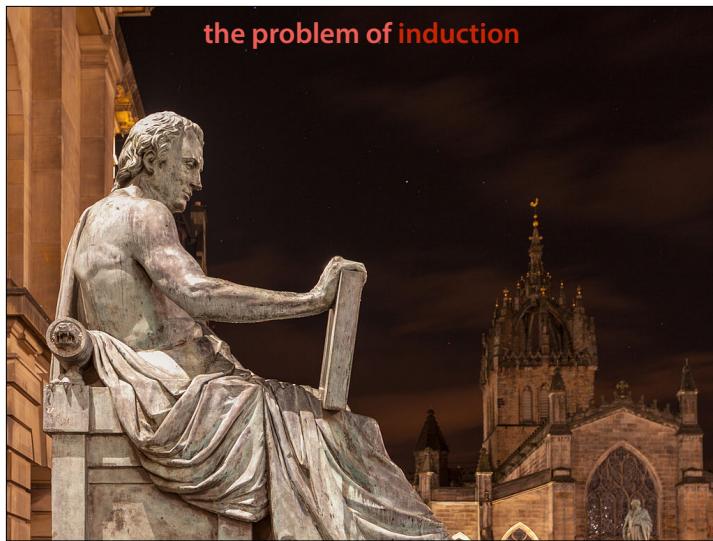
In order to pass the course, you need

- Pass the **exam**, or the resit, with a mark of at least 5.0.
- Complete a machine learning **project**. The mark should be higher than 4.5 and the assignment should be submitted by the deadline.

Read this page from top to bottom before you ask me any questions.

## what to do today

- Register for a workgroup (optional)
- Get a group together, or register for a random project group.
- First worksheet
- First homework



## the problem of induction

David Hume 18th century

Image source: By Bandan - Own work, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=29332424>

deductive reasoning	inductive reasoning
All men are mortal Socrates is a man	The sun has risen in the east every day of my life
therefore Socrates is mortal	so it will do so again tomorrow
discrete unambiguous provable known rules	fuzzy ambiguous experimental unknown rules

16

Inductive reasoning is essentially a philosophical name for learning. We observe something happening a number of times, so we *infer* that it'll probably happen again the next time. We're not absolutely certain, and it doesn't follow logically, but we're sure enough to use that knowledge to our advantage.

Deductive reasoning is rule-following. It's what computers do best. In order to make computers do something like inductive reasoning, and in order to fully understand how we do it, we need to reduce it to rules. But Hume argued that inductive reason can not be proved to work by deductive methods.

For deductive reasoning, we know the rules, and we understand them perfectly. For inductive reasoning the rules are not so clear. For instance, whenever I visit a funeral, I'm never the person being buried. Therefore, the more funerals I visit, the more certain I should be that next time it won't be my funeral. Clearly this is not the case (usually the opposite is true).

So, if inductive reasoning doesn't follow as a special case of deductive reasoning, and inductive reasoning applies sometimes and it doesn't at other times... how do we do it? Why is the funeral example obviously wrong, and the sun example obviously right? If inductive reasoning cannot be reduced to deductive reasoning, do we have any hope of reducing it to a computer program?

## a broad definition

(from expertsystem.com)

Machine learning [...] provides **systems** the ability to automatically learn and improve **from experience** without being **explicitly programmed**.

Here is a decent definition of Machine Learning that covers most of the important aspects. It's a system (i.e. a computer), it improves its behaviour based on experience, and the resulting behaviour has not been explicitly programmed.

Before we look at what this means in practice, let's have a look at some basic questions about machine learning:

- Where do you use it?
- When do you use it?
- What does it solve?

*quote source: <http://www.expertsystem.com/machine-learning-definition/>*

17

## where do we use ML?

### Inside other software

Unlock your phone with your face, search with a voice command,

### In analytics/data mining

Find typical clusters of users, predict spikes in web traffic

### In science/statistics

If any model can predict A from B, there must be some relation

18

## what makes a good ML problem?

- We can't solve it explicitly.
  - But approximate solutions are fine.
  - Limited reliability, predictability, interpretability is fine.
  - Plenty of examples available to learn from

bad	good
Computing taxes	Recommending a movie
Clinical decisions	Clinical decision <i>support</i>
Parole decision ( <i>support</i> )	Predicting driving time
Unlocking phone	Recognising a user

Fun fact, ML systems exist for parole decisions, and face detection is used for unlocking many phones. These are very problematic developments.

## what problems does ML solve?

We don't know a good explicit solution. But why not?

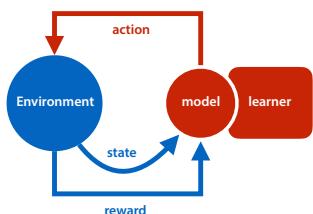
- Explicit solutions are expensive.
  - The best solution may depend on the circumstances.
  - The best solution may change over time.
  - We don't know exactly how our actions influence the world.
  - We can't fully observe the world.

Machine Learning may solve any or all of these problems.  
Depending on which problems you want to solve,

## the broad view: an intelligent agent

**online learning:** acting and learning at the same time.

**reinforcement learning:** online learning in a world based on delayed feedback.



If you have all these problems, i.e. you want to train a self-learning agent that continuously adapts and can freely move throughout a world, based on a sparse reward signal, you will need a very broad framework to model the problem. In **online learning**, we are choosing actions based on inputs, and also learning from those inputs at the same time.

In **reinforcement learning**, things are made even more complicated: we interact with a world that also gives us a reward signal. We try to maximise the reward, but we don't know which action leads to which reward.

Dealing with all these problems at once is very complicated. In most cases, we don't actually need to learn online. And instead of a reward signal, we have explicit examples of which input maps to which output. In such cases, we can use **offline learning**.

## offline learning

Separate *learning* and *acting*

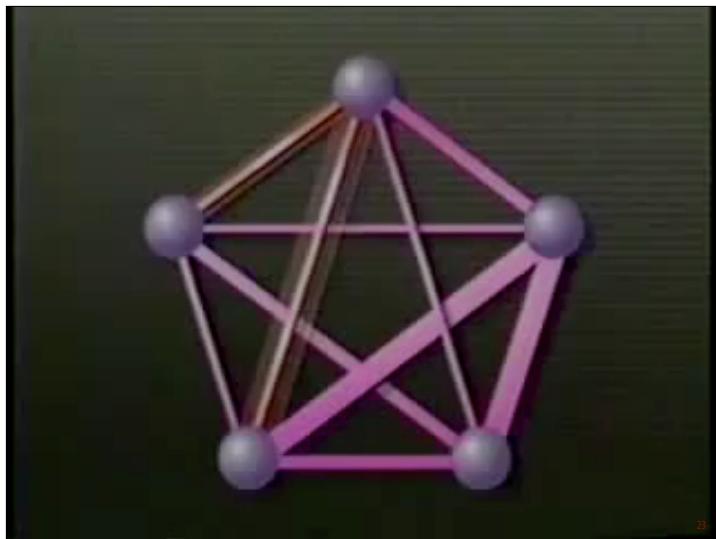
- Take a fixed dataset of examples (aka *instances*)
- Train a model to learn from these examples
- Test the model to see if it works

most of the course

In **offline learning** you separate the learning of a model and the use of a model in two separate steps. You gather a dataset of examples beforehand, you train a *model*, test it, and once you're sure it works well enough, you use that version of the model (for instance by sticking it into a computer program). The finished program will never learn while it's running. There is no feedback loop between learning and running.

While this robs the exercise of some of its more exciting aspects (it's a far cry from building androids), it still allows us to do something very useful: **it allows us to learn programs that we have no idea how to write ourselves**. For instance, we can learn a program that detects birds in pictures: we have no idea what kind of rules would be required, or how to design an algorithm to do it. Machine learning allows us to create such a program from a set of examples.

Almost all of this course will focus on offline learning.

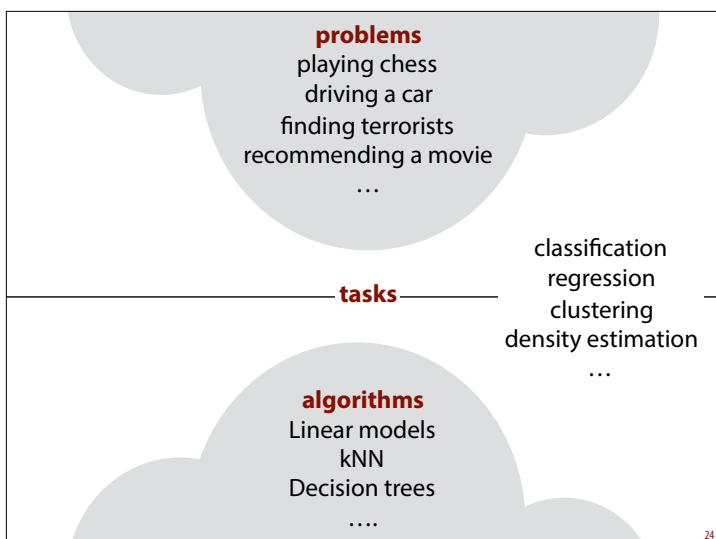


This video illustrates the basic idea: feed the computer the examples one by one, and tell it the target value for each.

*video source:*

<https://www.youtube.com/watch?v=7BtLqqJVP9w>

[https://archive.org/details/perceptron\\_documentary\\_excerpt](https://archive.org/details/perceptron_documentary_excerpt)



Now, the main problem with machine learning is that we want solutions that are applicable across domains. You don't want to dedicate your entire life crafting a perfect computer chess program, and then find out that your ideas have no use for anything else. We want to solve the problem of machine learning **in general**: instead of studying each problem in isolation, we want solutions that can be applied to many problems.

To make this possible, machine learning is usually built on one of a few **abstract tasks** like classification, regression or clustering. If you have a practical problem, like chess playing, you find a way to abstract the problem of playing chess (or part of it) to the generic task of, say, classification, and you can apply any existing machine learning problem.

## abstract tasks

### supervised

Explicit examples of input and output.

Learn to predict the output for an unseen input.

### unsupervised

Only *inputs* provided.  
Find any pattern that explains something about the data.

Abstract tasks come in two flavours: **supervised**, and **unsupervised**.

In supervised tasks, we have explicit examples of both inputs and the corresponding outputs. What we have to learn is the program that maps any input to the corresponding output. For instance, we may be provided with emails and given a label **spam** (advertising) or **ham** (genuine) for each. The task then, is to train a program to assign these labels to new emails.

In unsupervised tasks, there is no target value, only the data. All we can do then is to learn some structure in the data. For instance we can cluster students to see if there are natural groups, or see if we can detect which financial transactions are “unusual”.

25

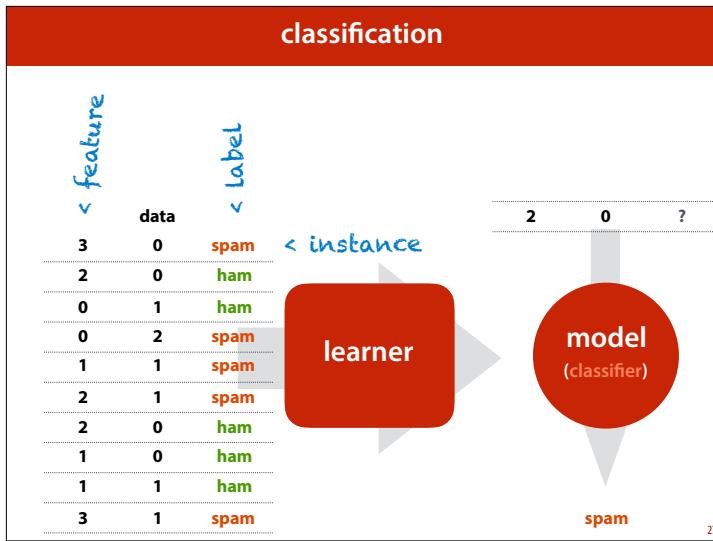
## supervised learning tasks

**Classification:** assign a class to each example.

**Regression:** assign a number to each example.

These are the two main forms of **supervised offline learning**.

26

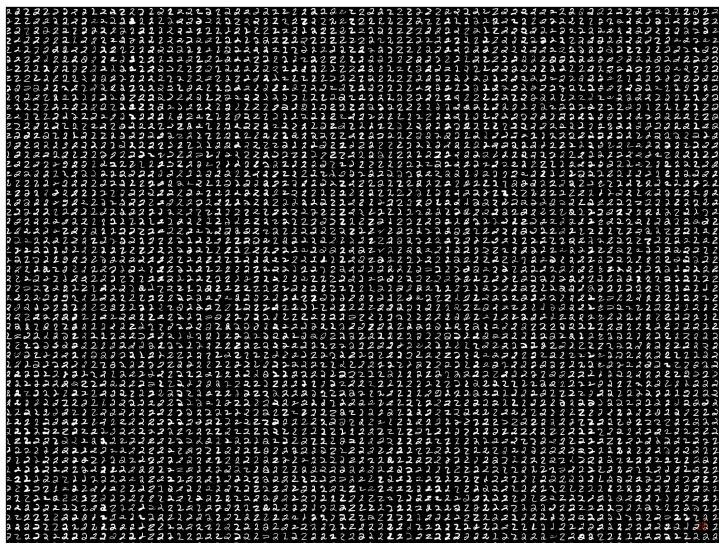


This is the basic framework of classification. The data that we provide our system with consists of examples, called **instances**, of the things we are trying to learn something about. In this example, our instances are e-mails.

We must then make a series of measurements about each instance. In the case of e-mails, we may measure how often a specific word occurs. The things we measure are called the **features** of the instance. We can measure numeric features (like age or speed), but they can also be categoric (like gender or color).

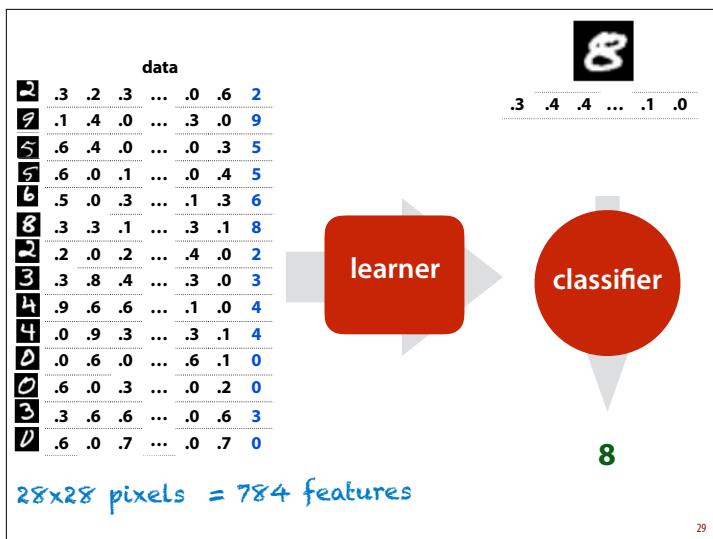
Finally we have the **target value**: the thing we are trying to learn. In classification, this is always a categoric value, or a *class*: one of a handful of possible values. In this case, is the e-mail **spam** (an unwanted advertising e-mail), or **ham** (a genuine e-mail).

This dataset is then fed to a learning algorithm. This can be anything, but it has to produce a classifier. A classifier is a small “machine” that (attempts) to solve the learning problem. That is, it takes a new instance, one that wasn’t in the original dataset, and for which we don’t know the target value, and it makes a guess at the target value.



Let's look at some examples of how we can reduce real-world problems to classification. We'll start with **handwriting recognition**.

This is a small selection from the MNIST dataset: 60 000 examples of handwritten digits. This translates very simply to classification: each picture of a digit is an **instance**, and the **target** is one of ten classes: 0, 1, 2, 3, 4, 5, 6, 7, 8 or 9.



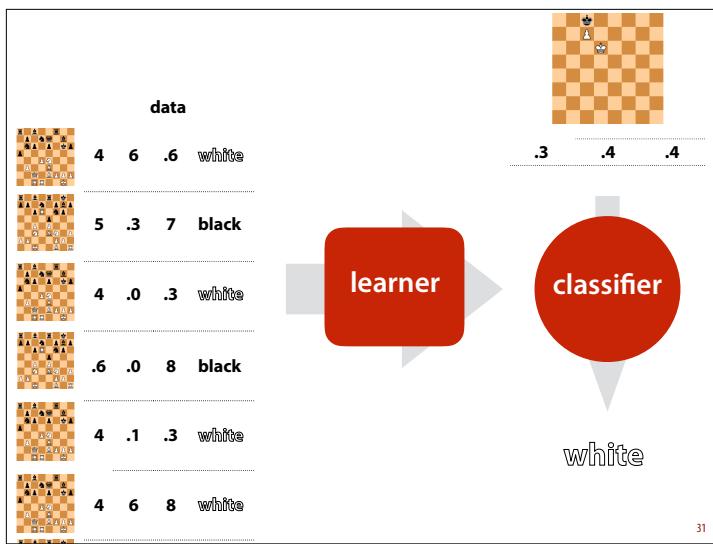
A simple way of attacking this problem is to make each pixel a feature. Here's what that looks like. For each instance, we translate each pixel to a value between 0 (black) and 1 (white). This gives us instance with 784 features each, labeled with a digit from 0 to 9. We feed these to the learning algorithm, which produces a classifier. We then get a new example, and ask the classifier what it thinks. Once we have a classifier that does well, we can use it in a larger system, for recognising digits.

The current best performing classifier for this task has a **probability of 0.21%** of getting an unseen example wrong.

Note that we haven't fully solved the problem of character recognition. We still need to cut a sequence of digits into individual digits, feed those digits to the classifier and process the results. This all the work we have to do to translate our **real problem** to the **abstract problem** of classification,



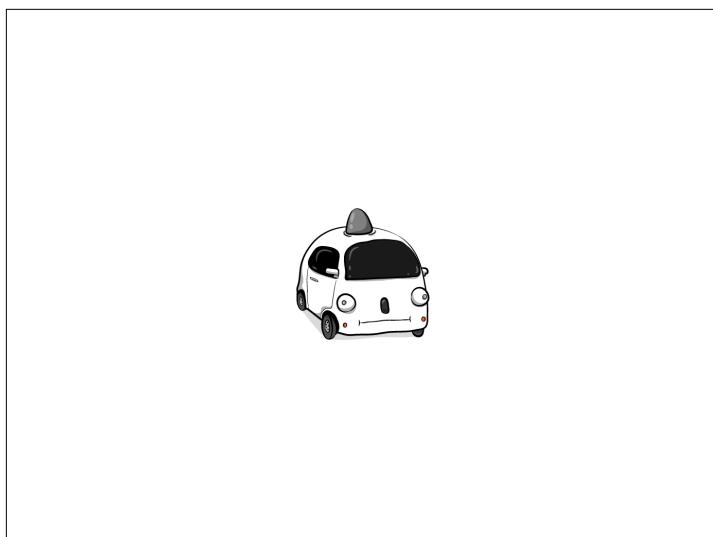
Let's look at a problem that's a bit further removed from classification: game playing. Specifically, playing chess. How do we abstract the problem of playing chess to a classification problem? The trick here is to make things easy for ourselves, by only abstracting part of the problem. We won't solve the whole thing with machine learning, but we'll learn a function that'll be useful.



For instance, we take a database of chess games, and to label each position with the winner of the corresponding game. The aim is to predict, for given a position, which player is going to win the game.

Again, if we solve this classification problem well, it doesn't give us a chess player. But a tree search chess algorithm (like minimax) could use such a classifier as a heuristic.

A difficult problem here is *which features to use*. One option is to report how much of each black and white piece is left, which would allow at least some positions to be predicted accurately, if one player has a strong material advantage. For more insightful learning, we need better features. **Domain expertise** can often be translated to good features: are there passed pawns, rooks on an open file, does a player own both bishops, etc.



Last example, a self driving car. How do we turn the problem of making a self driving car into a classification?

*image source: the oatmeal, [http://theoatmeal.com/blog/google\\_self\\_driving\\_car](http://theoatmeal.com/blog/google_self_driving_car)*

## ALVINN (1995)



960 features

33

Here is an actual self-driving car system from 1995. They used a very low-resolution, black-and-white camera to film the road, and observed a human driver's action to label each frame with an action. As with the digit recognition example, we simply make each pixel a feature.

This system actually drove from coast-to-coast autonomously in America (albeit with a human driver executing the system's instructions). NB: The actual system had more than three actions to allow for more gentle steering.

**problems**  
playing chess  
driving a car  
ocr

**classification**

**algorithms**  
?

This is what I meant by translating problems into tasks. By translating all these problems into classification problems, we can now apply any **classification algorithm** and see how well it does.

So how do we fill in the other half of this picture? Once we have a classification task, with features selected and a set of good examples, how do we actually produce a classifier?

We'll look at three simple examples: a **Linear classifier**, a **Decision Tree classifier** and a **Nearest Neighbors classifier**. We'll only explain them briefly to give you a sense of how these problems might be solved. Don't worry if you don't totally get it yet. All methods will be discussed in more detail in later lectures.

34



181	46	male
181	50	male
166	44	female
171	38	female
152	36	female
156	40	female
167	40	female
170	45	male
178	50	male
191	50	male

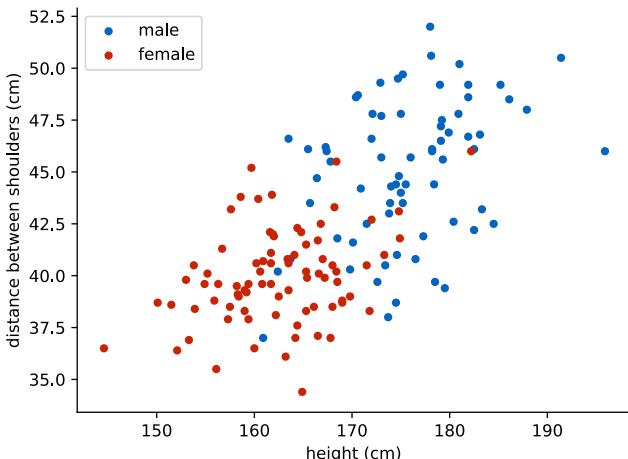
35

Here's a dataset that we'll use as a running example. Its *instances* are US soldiers, the two *features* are the height (or 'stature') and the distance between the shoulder blades ('interscye'). The class is their biological sex. Are these two values enough to predict whether somebody is **male** or **female**?

*data source: ANSUR II*

*image source: MEASURER'S HANDBOOK: US ARMY AND MARINE CORPS ANTHROPOMETRIC SURVEYS, 2010-2011*

feature space

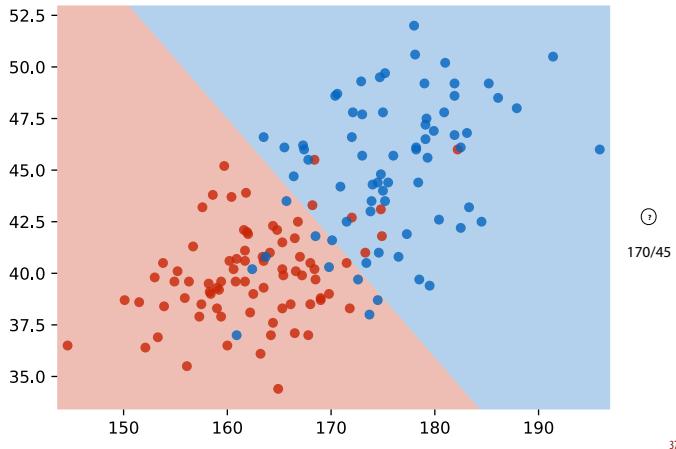


36

Since we have only two features, we can easily plot our dataset.

We call this space, where every instance is a point, the **feature space**. If we had 3 features, it would be a 3D space. For higher numbers of features, we may have difficulty visualizing the feature space, but that shouldn't stop the classifier: any classification method we come up with should work on an arbitrary number of features.

## example 1: linear classifier

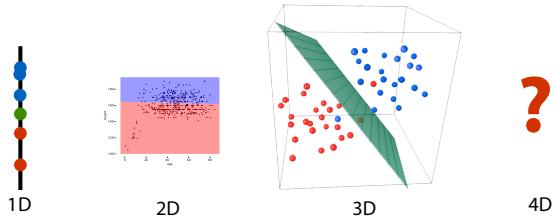


Here is a simple idea: *draw a line*. We just draw a line, and call everything above the line **male**, and below it **female**.

As you can see, a few examples end up misclassified, but most of them are on the correct side of the line. Our classifier would already do much better than one that would simply guess at random or call everything **female**.

If we see a new person, all we need to do is measure them, and see whether they end up above or below the line.

## hyperplanes



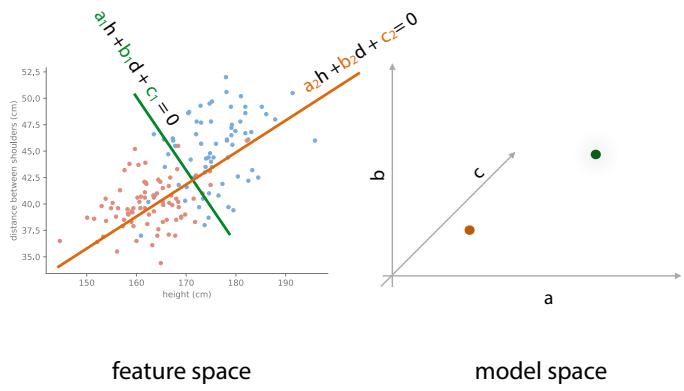
An important thing to note is that “drawing a line” is a technique that only works in two dimensions (i.e. if we have two features). Our methods need to work, at least in principle, for whatever number of features we decide to use. The more generic version of the idea to “draw a line” is to cut the feature space in two using a line-like shape.

In 1D, the equivalent structure is a point. Anything above the point is male, anything below it, female.

In 3D, we can cut the feature space in two with a plane.

In four or more dimensions, we can no longer draw it intuitively, but luckily the mathematics are very simple. We'll see how to define this in the next lecture.

## the two spaces of machine learning



Which line should we choose? We can visualise this problem in the feature space. In the feature space (or instance space), each instance is a point, and our current classifier is a line.

Since a line is defined by two parameters (the slope  $s$  and the intercept  $b$ ), we can visualise the **model space** as a plane. To search the model space, we define a **loss function** which tells us how well a particular model does for our data.

## loss function

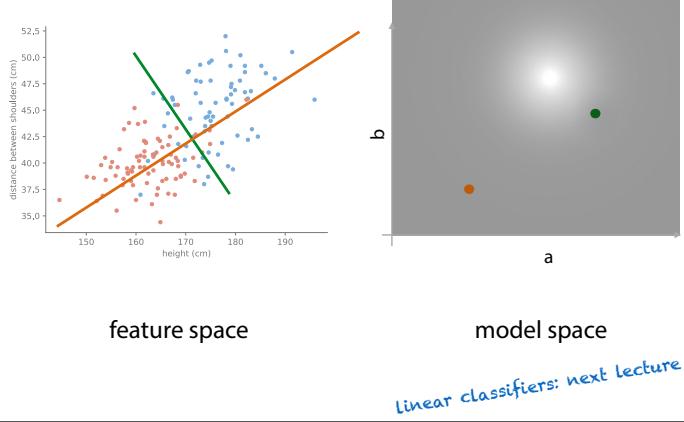
$\text{loss}_{\text{data}}(\text{model})$  = performance of **model** on the **data**  
(the lower the better)

for classification: e.g. the number of misclassified examples

Note that the loss function has the **model** as input and the **data** as a constant.

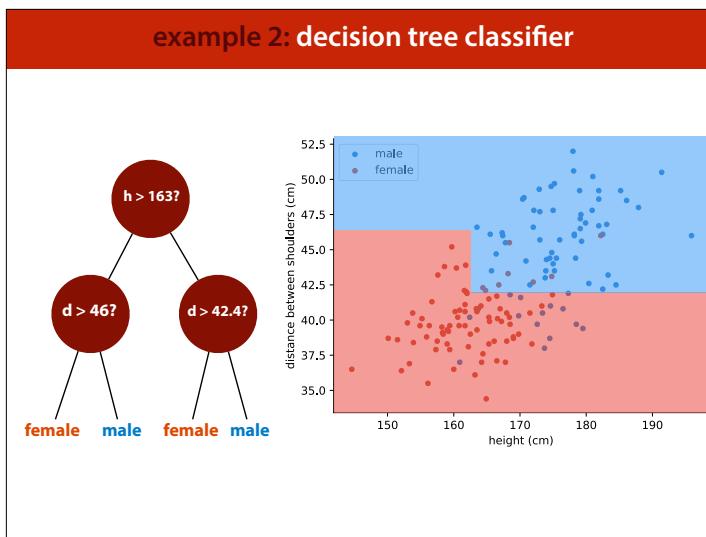
The best loss function to use for classification is a complex question. We'll come back to that later.

## the two spaces of machine learning



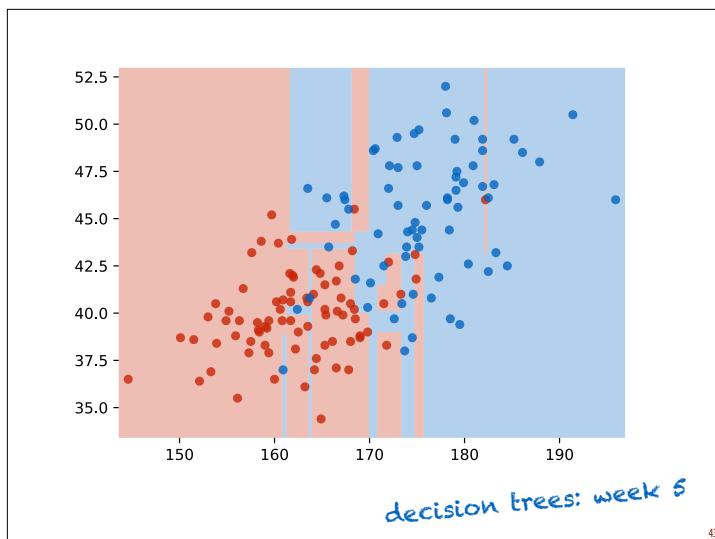
Once we have a loss function, we can colour our model space with the *loss of each model* (for our current data). The brighter, the better.

All we need to do now is find the brightest point, which corresponds to the best model. More on that next lecture.



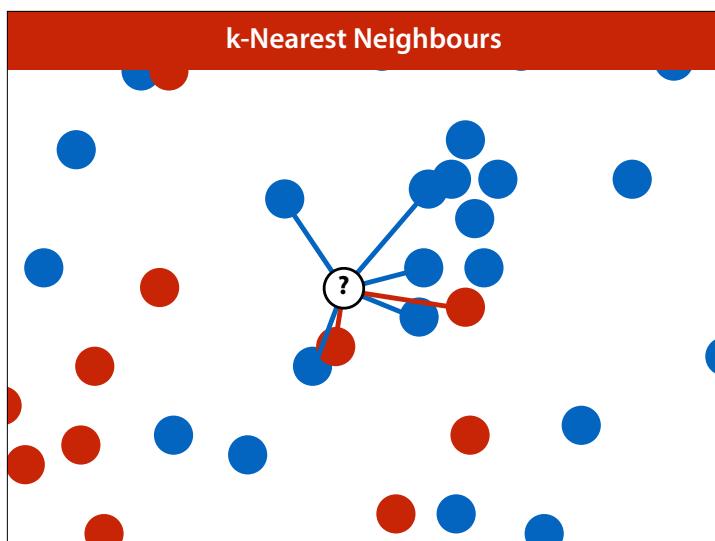
Here is a radically different approach: **a decision tree**. This classifier consists of a tree, which studies one feature in isolation at every node. In this case, it moves left if the feature is lower than some threshold value, and right if the feature is higher.

The model space of all possible decision trees is a lot more difficult to visualize. Often, decision trees are “grown” by adding nodes from the root until a particular criterion is reached. We’ll discuss how to train decision trees in detail in week 5 (the algorithm is simple, but it requires a little probability theory).



Here's what the actual decision tree algorithm from sklearn does (with default settings).

In week 5, we'll see how this learning algorithm actually works.

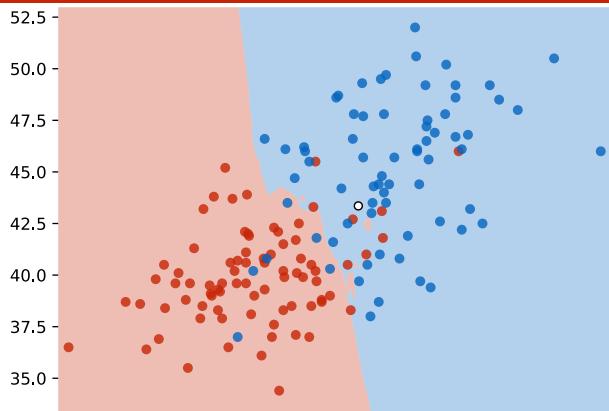


Finally, here is an example of a **lazy classifier**: k-Nearest neighbours. It doesn't do any learning. It just remembers the data.

For a new point (indicated by the question mark), it just looks at the  $k$  points that are closest ( $k=7$  in this picture), and assigns the class that is most frequent in that set (blue in this case).

$k$  is what we call a **hyperparameter**: you have to choose it yourself before you use the algorithm. We'll discuss how to choose hyperparameters next week.

## k-Nearest Neighbours



Here's what the decision boundary looks like for  $k=7$ . The point from the previous slide is indicated in white.

## variations

- Features: usually numerical or categorical.
- **Binary classification:** two classes
- **Multiclass classification:** more than two classes
- **Multilabel classification:** none, some or all classes may be true
- **Class probabilities/scores:** the classifier reports a probability for each class.

A few variations are possible on this basic scheme. Firstly, the features are usually numerical or categorical. Some models can handle only numerical, in which case, any categorical features have to be translated to numerical ones (we'll see how to do that next week).

**Binary classification** (a task with two classes) is probably the simplest and most well-studied task. If you have more than two classes, some classifiers (like kNN) can deal with that without a problem. For others (like linear classification), you'll need to find clever way to turn a binary classifier into a multiclass classifier.

**Multilabel classification** is a much more complex task. We won't go into it in this course, but it's an active subject of research.

Instead of a single verdict, it can often be helpful if a classifier assigns a **score** to each class. If we want a single class, we pick the one with the highest score, but we can also check what the second most likely class is, and how sure the classifier is of its verdict. This is often important if the consequences of a wrong classification are very serious (i.e. deciding whether to operate on someone, or whether to investigate someone for criminal activity).

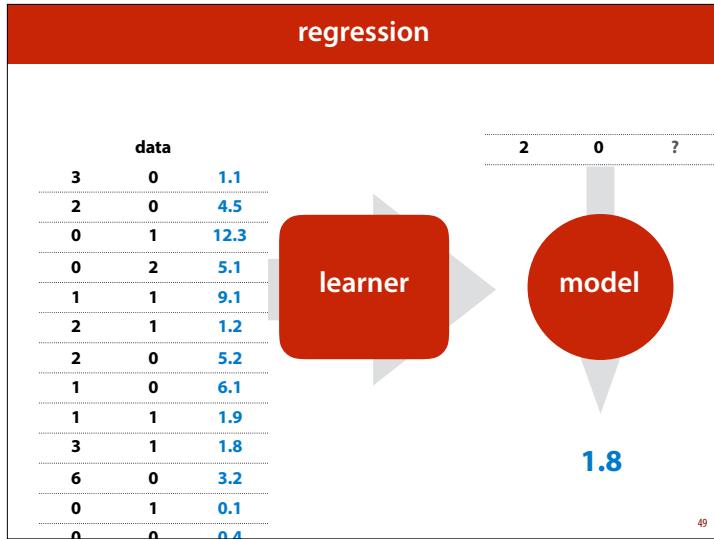


image source: <https://twitter.com/archilllinks/status/1022889384494940160>

---

regression

## regression



But classification isn't the only abstract task you can reduce your problem to. If your target value is expressed as a numeric value, we call it **regression**. It works exactly the same as classification, except we're predicting a number instead of a class. That is, the model we're trying to learn is a function from the feature space to  $\mathbb{R}$ .

*feature* >

< *target*

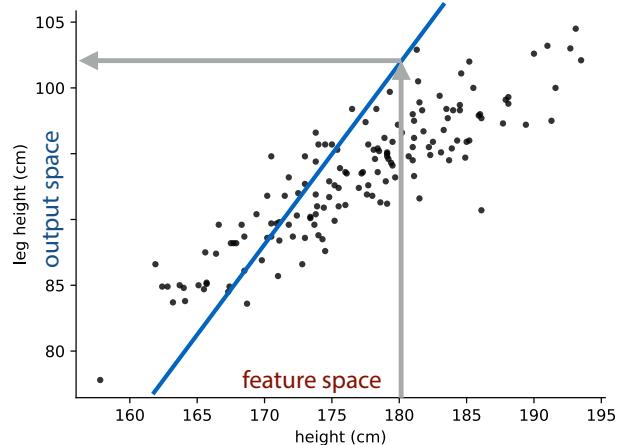
174	96
181	98
177	93
178	92
178	92
181	100
186	98
179	94
174	92
175	94

50

Staying in the same domain, here is another dataset. Note that we have **only one feature** this time. The other numerical column is the **target** data. For general regression we should be able to handle datasets with arbitrary numbers of features.

*data source: ANSUR II*

*image source: MEASURER'S HANDBOOK: US ARMY AND MARINE CORPS ANTHROPOMETRIC SURVEYS, 2010-2011*



Here's what our data looks like. Note that though it looks the same as in the classification example, this time we're plotting both the outputs and the **feature space** in the same figure.

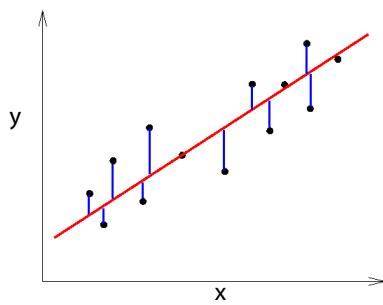
We can use a linear model again. But note how differently we're using the model. Previously, we wanted to segment the feature space into two classes. Now we're trying to model the relation between the feature(s) and the target.

Now, the line I've drawn here isn't very good. It predicts much too high a value. To determine how good a model is, we must choose a **loss function**.

### loss function for regression

$$\text{loss}(\mathbf{p}) = \frac{1}{n} \sum_i (f_{\mathbf{p}}(x_i) - y_i)^2$$

aka the **mean-squared-errors** (MSE) loss.



The **loss function** maps a model to a number that expresses how well it fits the data (the smaller the loss, the better). Offline machine learning boils down to finding a model with a low loss for your data.

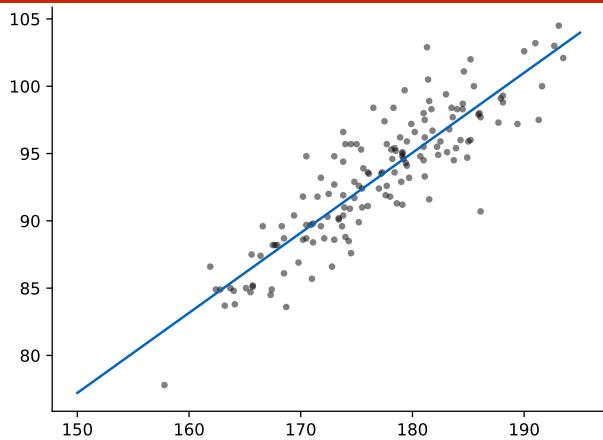
Here is common loss function.  $\mathbf{p}$  stands for the parameters that define the line. We simply take the difference between the model prediction and the target value from the data. This is called a **residual**. We square, and then sum all residuals.

Sometimes we also divide by the size of the data ( $n$ ). Occasionally, we multiply by  $1/2$  for technical reasons. This changes the actual value, but not which model produces the lowest loss, which is the question we want to answer.

*image source: <http://cs.wellesley.edu/~cs199/lectures/35-correlation-regression.html>*

## linear regression

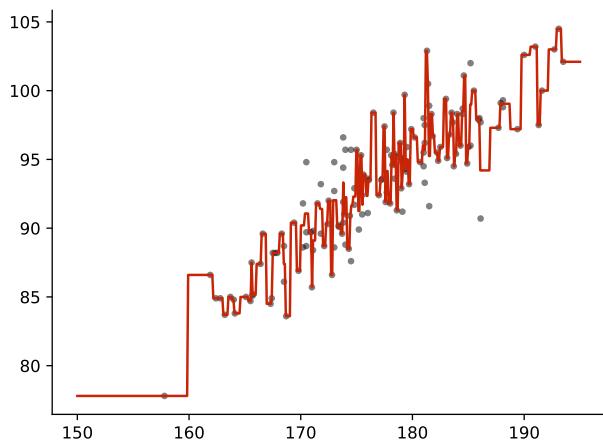
This is the line with the lowest MSE loss for this data.



53

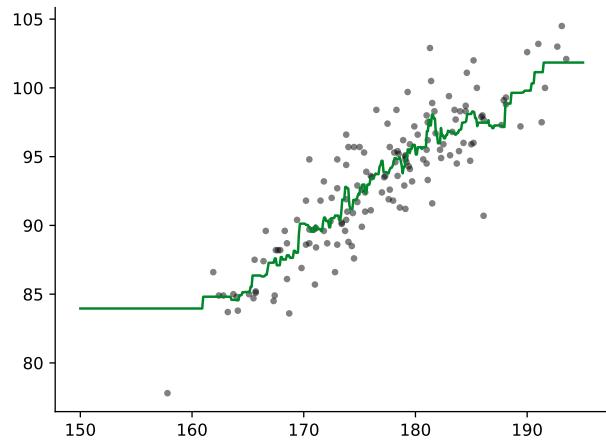
## Regression tree

We can also use the **decision tree** principle to perform regression. We simply segment the feature space into blocks, using a tree as before, and instead of assigning each a *class*, we assign each a *number*. This model covers the data very well, for most points in our data it predicts exactly the right value.



54

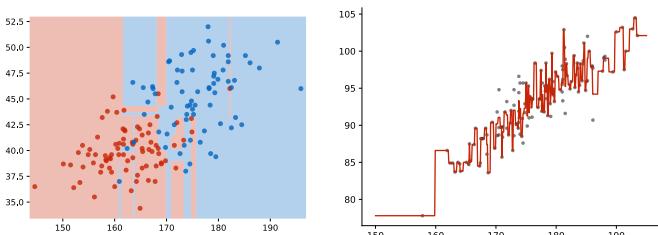
## kNN regression



For the sake of completeness, here is what kNN regression looks like. Its prediction is the average of the  $k$  nearest points in the data ( $k=7$  here).

55

## Overfitting



These models achieve much lower loss (Accuracy/MSE) than the linear or kNN variants. **Would you use them?**

Bear in mind that you usually *can't look at your model like this*. You'll often have hundreds of features, and all you'll have to go on is the **training loss**: the loss for a given model on the training data.

56

# Never judge your performance on the training data

This is the most important rule in machine learning. How many of the training instances the model gets right means nothing. You need to hold back some data that the model hasn't seen during training, and judge the performance on that.

## split your test and training data

**training data**      **test data**

- The aim of machine learning is not to minimise the loss on the **training data**, but to minimise the loss on your **test data**.
- You don't get to see the **test data** until you've chosen your model.

This is what we call the test set. We withhold some random instances and see how many the classifier gets wrong, once it has finished training.

## in code (cf worksheet 2)

```
# Choose a model: Decision Trees
from sklearn.tree import DecisionTreeClassifier

x_train, x_test = ... # matrix of features
y_train, y_test = ... # target class labels

tree = DecisionTreeClassifier()
tree.fit(x_train, y_train) # Search for model

# classify new data
y_predicted = tree.predict(x_test)
# see how well we do
print accuracy_score(y_predicted, y_test)
```

Here's what the basic recipe looks like in code (using the `sklearn` library). Note that the actual machine learning happens in just two lines of code.

All you need to do is decide your features, and decide your target values (classes in this case). Once you've done that, you're doing machine learning in two lines of code. You can then test how well your model does (more about that in week two) and keep trying different models until you get the performance you're happy with.

59

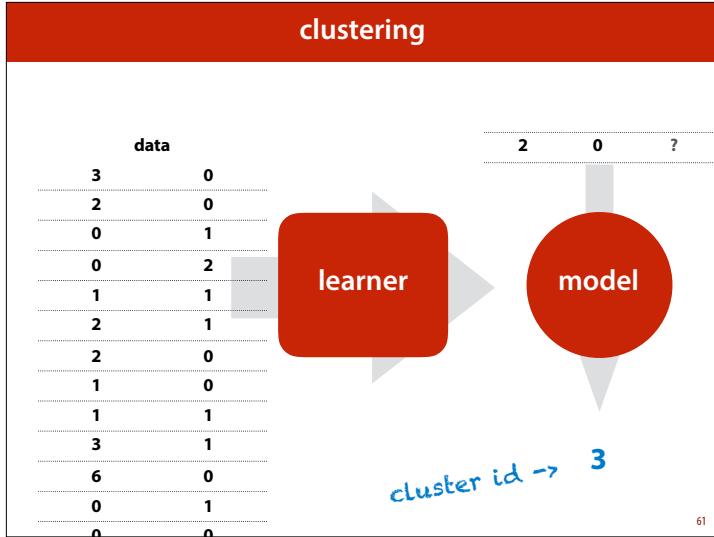
## unsupervised learning tasks

- Clustering
- Density Estimation
- Generative Modeling

In regression and classification we had a set of examples: we had **labeled data**. In other words, we could show our learner exactly what we want it to learn. In **unsupervised learning**, we don't have that luxury. We just have the data, and we want the learner to expose some kind of structure in it.

60

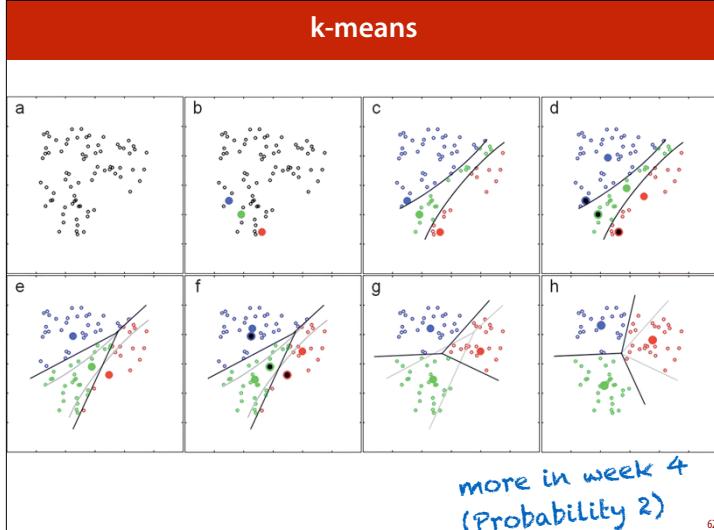
## clustering



In the case of clustering, we ask the learner to split the instances into a number of clusters. The number of clusters is usually given beforehand by the user.

You can think of this as unsupervised classification.

## k-means



Here is one example of a clustering algorithm.

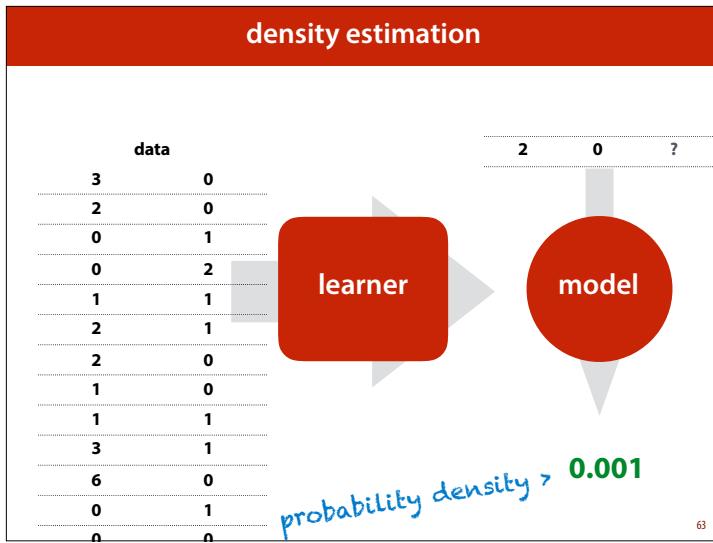
This algorithm is called k-means (not to be confused with k-NN). In the example we will separate the dataset shown in (a) into three clusters.

We start (b), by choosing three random points in the feature space (the red green and blue points), called the "means". We then colour every point in the colour of the nearest mean.

The second step is to remove the original means, and recompute them as the actual means of the sets of red, green and blue points. We then iterate these two steps, alternately recomputing the means and recovering the points, until the algorithm converges to a stable state.

image source:

## density estimation

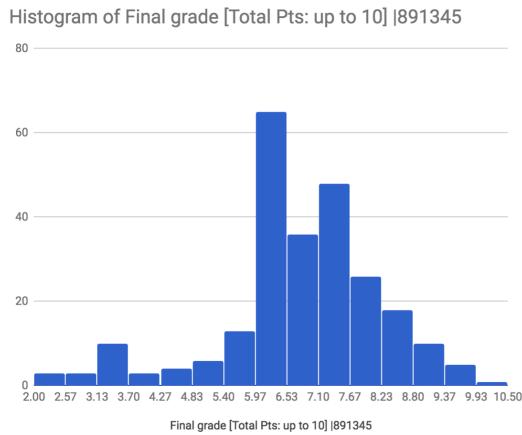


In density estimation, we want to learn how *likely* new data is. Is a 2 m tall 16 year old more or less likely than a 1.5 m tall 80 year old?

The number that the model produces should be a *probability* if the feature spaces is discrete. That means that the sum of all answers over the whole feature space should be 1.

If the feature space is continuous (numeric features), the answer should be a probability *density* (and all answers should *integrate* to 1).

## density estimation



Density estimation is the task of modelling the *probability distribution* behind your data.

Here is an example: the final grades from 2017. If you know a bit of statistics, you can probably see sort of a normal distribution in this. Once you've fitted a normal distribution, you can give a density estimate for any grade.

Except that there are three peaks. These could be explained by noise, but we could also fit a mixture of three normal distributions to this data, to explain the peaks. This is a much more difficult task, to which we will return in a few weeks.

## Generative modelling (sampling)

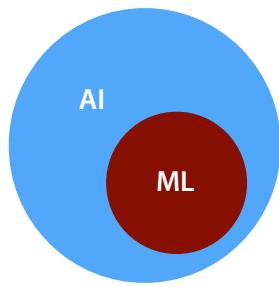


With complex data, it's often easier to *sample* from a probability distribution than it is to get a probability (density) estimate. Building a model from which you can sample new examples is called **generative modelling**.

These people *don't exist*. These pictures were *sampled* from a model trained on a large dataset of images of faces. Note that this is not a 3d model, or a generator that started with a basic face and filled in the details: all the model saw was a large collection of pictures. This is a typical example of the power of **deep learning**, which we will discuss in the third week.

*image source: <https://arxiv.org/abs/1812.04948>*

## Machine Learning vs. AI



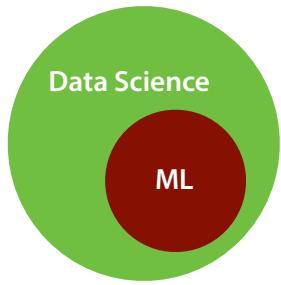
**AI, but not ML:** playing chess, automated reasoning, planning.

I've told you what machine learning is. Now let's look at what it isn't. How it relates to other fields and what the differences are.

First up, Artificial Intelligence. Machine Learning is a subfield of AI. If we want to make a general AI that can do everything we can, it needs to be capable of learning. But there are many other problems and fields in AI that have nothing to do with learning.

Other subfields, like natural language processing, are greatly helped by ML techniques, but can also be tackled without.

## Machine Learning vs. Data Science



**Data Science, but not ML:** Gathering data, harmonising data, interpreting data.

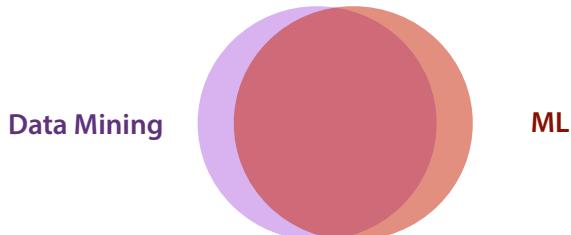
next week

Similarly all ML is Data Science, but not all Data Science is ML. Often, ML is used as part of a larger data science pipeline.

Thursday next week, we will look at some of the data science aspects required to perform Machine Learning experiments.

67

## Machine Learning vs. Data Mining



**More DM than ML:** Finding common clickstreams in web logs. Finding fraud in transaction networks.

**More ML than DM:** Spam classification, predicting stock prices, learning to control a robot.

68

Here is a more subtle distinction: data **mining** and machine learning. Opinions differ, but I would characterise them as follows. Both analyze data, but they do so with different aims: machine learning aims to produce a *model* of the data, data mining aims to produce intelligence.

Another distinction is that machine learning focuses on prediction: trying to predict a target value for new data, whereas data mining tries to navigate and simplify the data so that it becomes useful for users.

If you have a dataset, but you expect never ever to see any new data from the same source, you can perform data mining on it, but performing machine learning on it is not much use (although your data mining will probably use machine learning *techniques*).

## Machine Learning vs. Information Retrieval

Information  
Retrieval

ML

Information retrieval (building search engines) may look at first like a field that is completely distinct from ML. But on closer inspection, it turns out that you can model IR as a kind of classification task: your instances are documents, and your aim is to classify them into relevant or irrelevant (for a particular query).

This may seem a bit extreme, since the class imbalance is so high, but this has actually helped us in ML to think more clearly about problems with high class-imbalance (where ranking is a more appropriate way to think about the task than classification).

69

## Machine Learning vs. Statistics

Statistics

ML

**Stats but not ML:** Analyzing research results. Experiment design. Courtroom evidence.

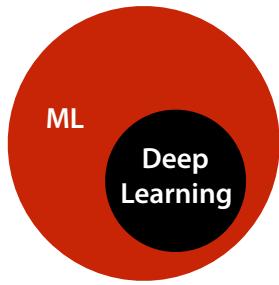
**More ML than Stats:** Spam classification, movie recommendation.,

Here is another subtle distinction. Statistics and ML both focus on analysing data, and modelling it in some way. In fact, many of the models we use in ML were invented by statisticians before the name Machine Learning existed. The distinction isn't always clear, but the most important difference is that Statistics aim to get at the *truth*, whereas machine learning tries to come up with something that *works*, regardless of whether it's true.

Consider spam classification. We usually model emails as a bag of independently drawn words. This has nothing to do with the way emails are actually written. Still, it works well enough to let people control their in-box. Contrast this with proving in a courtroom that a particular piece of DNA evidence really puts a suspect at the scene of the crime. Here, we're interested in more than just getting a useful model that captures some of the data, we want the truth.

70

## Machine Learning vs. Deep Learning



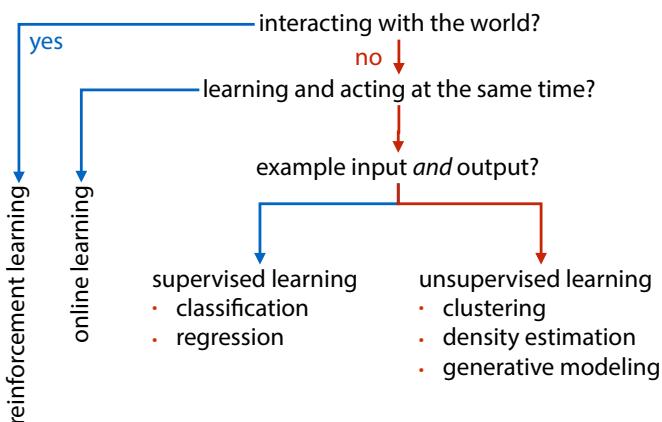
starting in  
week 4

Finally, you may have heard a lot in the news in recent years about deep learning and wondered whether it is the same as machine learning or something different, or what. Here I can be clear. Deep Learning is a subfield of Machine Learning. All Deep Learning is Machine Learning but not all Machine Learning is Deep Learning.

We will discuss Deep Learning, and what makes it so special at various points in the course.

71

## summary



72

## offline machine learning: the basic recipe

**Abstract** (part of) your problem to a **standard task**.

Classification, Regression, Clustering, Density estimation, Generative Modeling, Online learning, Reinforcement Learning, Structured Output Learning

**Choose your instances** and their **features**.

For supervised learning, choose a target.

**Choose your model class**.

Linear models, Decision Trees, KNN,

**Search** for a good model.

Usually, a model comes with its own search method. Sometimes multiple options are available.

To summarize: this is the basic recipe for doing machine learning. It doesn't always fit the problem, and we'll look at those cases too. Sometimes your problem doesn't fit very well (reinforcement learning, recommendation), sometimes deviating slightly can help performance (sequence learning) and sometimes, taking a radically different approach can turn everything on its head (deep learning).

Nevertheless, if you want to keep things simple, and use existing solutions, **the basic recipe** is a good starting point,

73

## summary

**abstracting** your problem: **instances**, **features**, **target**.

**supervised** learning: classification, regression.

- linear models, tree models, NN models

**unsupervised** learning: clustering, density estimation, generative modeling.

**loss function**: maps a choice of model to a *loss* for the current data (the lower the better).

**overfitting**. Test and training data.

74

week 1	week 2	week 3	week 4
<b>Introduction</b> Classification, Regression, Clustering	<b>Method 1</b> Comparing methods <b>Method 2</b> preprocessing missing values, outliers, dim. reduction	<b>Probability 1</b> Gaussians, (Naive) Bayes Entropy, Logistic regression <b>Linear Models 2</b> Neural nets, SVMs	<b>Deep Learning 1</b> SGD Backpropagation CNNs <b>Probability 2</b> Expectation Maximization <b>!pick a topic!</b>
<b>Linear Models 1</b> Hyperplanes, Random Search, Gradient descent			
week 5	week 6	week 7	week 8
<b>Deep Learning 2</b> Generative models: GANs, VAEs	<b>Sequences</b> Markov models, Word2Vec, RNNs <b>Matrix models</b> Recommender systems	<b>Reinforcement Learning</b> Q learning, Policy gradients, <b>AlphaStar</b> <b>Review</b> Open problems The future of machine learning	<b>exam</b> 25 March <b>project deadline</b> 29 March

75

## what to do now

- Register for a workgroup (optional)
- Get a group together, or register for a random project group.
- First worksheet
- First homework

76

**HAPPY LEARNING!**

If you have any questions, please ask them on the Canvas discussion board if at all possible. If it's a personal question, or if you need to ask me personally for some other reason, please don't hesitate to email.

 mlcourse@peterbloem.nl

← go easy on me!